Threading

About Threads

- A **thread** is an execution context within the overall process
- Multiple threads can run simultaneously
  - Independent code location (program counter)
  - Shared memory, files, etc.
- Example: a book
  - everyone shares the overall book
  - each person has their own context – which page they're on
  - the book is passed back and forth, switching context each time, but the contents of the book don't change
  - notes made by one person are visible to all other readers

Android Threads

- Android apps use threads – you can assign work to a particular thread as appropriate
- Threads *are not* tied to Activities – they can last longer (or shorter)
- There are some specific named threads:
  - the Main Thread (often called UI Thread)
  - the I/O Thread, for input/output operations (internet, local files)
  - the Computation Thread, for long-running calculations
- It is also possible to create a new thread at any time
  - this may incur overhead penalties – consider using an existing thread!

Android Main Thread

- Responsible for everything that happens on the screen
  - drawing Views, processing user touch events, etc.
- The Android UI toolkit is *not* thread-safe!
  - thread-safety means running the same code simultaneously
- There are two rules to Android's single thread model:
  - Do not block the UI thread
  - Do not access the Android UI from outside the UI thread

Why Threads?

- Any long running tasks on the main/UI thread consequently block any other processes that run on that thread until the task completes
- In order to allow for smooth UI operation and also provide a mechanism for executing long running background tasks, use the provided alternate threads or create your own

Tasks Requiring Threads

- Asynchronous calls to remote services inherently have an unknowable runtime. Given that many contemporary applications have a dependency on remote services (for data, media, etc…) it is common for them to employ threading to run these operations without blocking the UI thread.
- Background calculations sometimes can be long running depending on the complexity of the input and translation to the expected output. In cases like this, a thread is appropriate.

Thread Pools

- In cases where more than a single background thread and a UI thread are required (several concurrent tasks running), a Thread Pool can be created.
- Thread Pools allow tasks to be queued up for execution once a thread becomes available.

Dangers of Threading

- The primary danger of threading is the introduction of race condition due to dependencies on resources to be returned by threaded execution.
- The best way to avoid this is to make sure that the handling for those dependencies occurs in whatever mechanism that you use to return the response to the UI thread.

Android Threading Mechanisms

- Runnables and Handlers
  - A more primitive approach using older Android technology
  - Offers very fine control
  - Logic for returning to the UI thread from a background thread must be implemented by the engineer
- AsyncTask
  - More contemporary technology
  - Offers a simplified interface for creating threads
  - Logic for returning to the UI thread from a background thread is already implemented as part of an abstract class

```java
public class AsyncTaskTestActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...

        new MyTask().execute("my string paramater");
    }


    private class MyTask extends AsyncTask<String, Integer, String> {

        @Override
        protected void onPreExecute() {

        }

        @Override
        protected String doInBackground(String... params) {

            String myString = params[0];

            int i = 0;
            publishProgress(i);

            return "some string";
        }

        @Override
        protected void onProgressUpdate(Integer... values) {

        }

        @Override
        protected void onPostExecute(String result) {
            super.onPostExecute(result);

        }
    }
}
```